Proceedings of ICAPP 2014
Charlotte, USA, April 6-9, 2014
Paper 14384

NEAMS MeshKit: Nuclear Reactor Mesh Generation Solutions

Rajeev Jain and Timothy J. Tautges
Argonne National Laboratory, Argonne, IL 60439
Tel: 630-252-3176, Fax: 630-252-5986, Email: jain@mcs.anl.gov

Abstract – *NEAMS MeshKit module is capable of generating geometry and meshes that accurately describe the computational model for simulating a variety of nuclear reactors. MeshKit uses a graph-based process for specifying the overall meshing problem, with graph nodes representing meshing and other operations, and graph edges as dependencies between those operations. In our previous papers on Reactor Geometry (and Mesh) Generation (RGG) tool in MeshKit: basic working, examples of models generated using serial and parallel version of the tool were presented. In this paper, we briefly describe the MeshKit design philosophy and list currently available algorithms for Hex/Quad, Tet/Tri along and other helper-meshing algorithms in MeshKit. New keywords and features introduced to RGG are highlighted with the help of examples. Reactor assembly examples include the XX09 assembly used in Shutdown Heat Removal Tests (SHRT) that demonstrated passive safety features of EBR-II Experimental Breeder Reactor and an Advance Burner Test Reactor (ABTR) fuel assembly. Reactor core examples include the ABTR core model with restraint rings, homogenized High Temperature Gas-cooled Reactor (HTGR) and homogenized EBR-II core model with detailed XX09 assembly.*

I. INTRODUCTION

Mesh generation is a challenging field of science that has made rapid advancement over the years. New discretization techniques, improvements in computer hardware and robust solver methods have driven the research in this field. While several open source and commercial mesh generation tools were already available when this project was started, each of these tools were deficient in some way, either being too restrictive in their licensing (e.g. GPL), focused on only one specific meshing technology (triangles, tetrahedral), or lacking support for other parts of the meshing process (e.g. geometric model query). MeshKit is meant to overcome these problems, as well as to serve as a delivery vehicle for specific meshing tools and algorithms for various applications. MeshKit uses a graph-based process for specifying the overall meshing approach, with graph nodes representing meshing and other operations, and graph edges as dependencies between those operations. This approach supports the traditional BREP-driven meshing process, but also supports more complex, multi-step meshing procedures, like post-mesh boundary layer generation, mesh generation then refinement, and assembly of large meshes using read/copy/move operations.

Preparation of a model suitable for analysis is an important step in any physics simulation. A nuclear reactor comprises of ducts, fuel rods, control rods, cladding, load pads, restraint rings, grid spacers, coolant flow regions, inter-assembly gaps and other instrumentation pieces. Detailed modeling of all the pieces and naming of material and boundary conditions in the core model is often very tedious and hard to perform manually. NEAMS MeshKit module is an effort to simplify and minimize the human interaction during this process. MeshKit is capable of generating geometry and meshes that accurately describe the computational model for heat-transfer, fluid mechanics, neutronics, structural mechanics and other simulations. Two key algorithms - AssyGen and CoreGen are termed as the RGG tool in MeshKit. These algorithms are presented in our previous paper titled “Creating Geometry and Mesh Models for Nuclear Reactor Core Geometries Using a Lattice Hierarchy-Based Approach” [1]. This paper cites the problems encountered during utilizing conventional meshing tools for modeling nuclear reactors. A survey of all existing tools is presented and a novel three-stage approach is described. A very few efforts have been found to develop solutions for automatic the reactor model generation. RGG is the first tool that is capable of generating assembly and core mesh models of hexagonal and rectangular reactors automatically. The three-stages workflow described in the paper is 1. AssyGen, 2. Meshing and 3. CoreGen. In AssyGen stage, reactor assembly geometry is generated and CUBIT [2] meshing script is created. Next, meshing of assembly geometry can be performed optionally using the CUBIT script generated in AssyGen stage or using MeshKit algorithms or using external meshing tools. Finally, the CoreGen program

creates the reactor mesh by arranging all the assembly and interstices meshes. This workflow allows a balance between automation and user-intervention; RGG supports both text file-based automation as well as manual intervention to add model details (e.g. grid spacers). Nomenclature for representation of full, $1/12^{th}$ and $1/6^{th}$ (hex-flat and hex-vertex) hexagonal reactor core lattice, along with rectangular lattice is described in the paper. CoreGen also has features for creating geometry only models. In another approach presented in the paper, 2D mesh creation and a special four-stage process is described, wherein a 2D mesh model for the core is generated, followed by extrusion into the third dimension. This approach requires about half the execution time of the original three-stage process, mostly because of a large reduction in the number of vertices being compared for merging. It is important to note that the nodes on faces of different assemblies that come in contact with each other must be matching for this process to work. This can be easily achieved for hex-meshed assembly meshes by setting same intervals on all side surfaces. For generating tetrahedral core meshes with this three-stage process the primary difficulty is the careful application of surface splitting, mesh reflection, and mesh copying, to ensure both translational and rotational symmetry of mesh on the side surfaces of assemblies. Propagation of mesh metadata during the core mesh generation (CoreGen) stage is described in detail. Several examples of hexahedral and tetrahedral core mesh generation for a variety of nuclear reactors are also presented. It is found that the total human time required for modeling a $1/6^{th}$ of a Very High Temperature Reactor (VHTR) core drops by a factor of 50 to 100. Meshes created by RGG have been successfully used by a variety of physics codes such as Nek5000 (open source CFD code), Diablo (Structural mechanics code), PROTEUS (Neutronics code) and STAR-CCM+ (black box commercial CFD code).

In a follow-up paper titled "RGG: Reactor Geometry (and Mesh) Generator" [3] algorithm and results of parallel version of CoreGen are described. The original three-stage approach for generating lattice-based models was extended in several ways from that reported in the previous paper. Parallel version of CoreGen allows the problem to fit in memory, allowing creation of large reactor meshes that would otherwise be impossible to create using conventional meshing tools. The central idea of this tool is based around distribution of copy/move and merge work among the processors utilizing the symmetric lattice-based arrangement of pins and assemblies that form the reactor core. Superlinear speedups are observed, thereby significantly reducing the total time required for generating large models. Several examples including VHTR models, a ¼ PWR reactor core, and a Full Core model for MONJU are reported.

In this paper, our objective is to describe the unique design of MeshKit and briefly describe Hex/Quad, Tet/Tri and other meshing algorithms available in MeshKit. In-order to utilize the current state-of-art and popular packages, MeshKit supports interoperability with other packages. We have interfaces to popular packages such as CAMAL, Triangle, NetGen etc. The graph-based design enables all these packages to interoperate and solve a particular meshing problem. In Section 2, we highlight all required and optional packages that MeshKit relies on for various functionalities. Section 2 also describes the design philosophy and overall organization of the library. New results from AssyGen and CoreGen are presented in Section 3 and Section 4 respectively. Section 3 and 4 also include the details of new features such as automatic creation of AssyGen input files based on different material, boundary conditions and assembly specification of the core. CoreGen's ability to create core geometry models has been utilized for the creation of interstices mesh that includes restraint ring in the ABTR core model. Finally, we present the conclusions and future work in Section 5.

II. MESHING ALGORITHMS

Most current meshing environments are targeted toward a Boundary REPresentation (BREP)-based approach, backed by a geometric model and associated topology (vertices/edges/faces/regions and adjacency relations between them). The meshing process usually proceeds by meshing BREP entities in increasing topological dimension, starting with vertices, then edges, and so on. However, this model is deficient in several ways. First, not every meshing process needs or has a geometric model representing the entire domain to be meshed. The best example of this is the RGG tool, where individual assembly types have geometric models but are then copy/moved into a lattice of assembly models forming a reactor core. Second, a meshing procedure may not involve only a once-through meshing of each BREP entity; again, RGG is a good example of this, where the first part of the process involves meshing BREP models, but the last step involves copy/moving mesh subsets into a larger core lattice. Finally, the procedure-driven approach to meshing represented by most CAD-based meshing tools fails to capture the parallelism and dependency structure that can be found in most meshing problems (including BREP-based ones); representing and exploiting this richer structure provides more flexibility while still being applicable to BREP-based problems.

MeshKit models the general meshing problem as a directed graph-based process, with graph nodes representing individual steps in the process and graph edges representing dependencies between those steps. For convenience, the graph always has a single root and leaf node, with one or more possibly-independent paths between them. Each graph node represents an explicit step in the meshing process, whether that involves generation of new mesh or performing some other operation on existing

mesh. The part of the model operated on by that operation is stored as input for that node, along with any control parameters specific to the operation. The meshing process is executed by traversing the graph twice, once in reverse direction (from leaf to root), to perform necessary setup actions and create upstream graph nodes not explicitly created by the application, then in forward direction, to perform the action represented by each graph node.

Fig. 1(a), (b) and (c) show the steps used for meshing of a geometric surface bounded by two edges using the graph-based approach. In this example the steps followed are similar to conventional BREP-based mesh generation. For the starting mesh graph, the user specifies a "trimesh" meshing operation, assigned to the surface, then the setup traversal in trimesh operation determines that it needs a mesh for the bounding edges, and creates a graph node representing that operation (an EqualEdgeMesh meshing operation applied to edges E1 and E2). Continuing the traversal, the setup operation on the EqualEdgeMesh node determines that the two bounding vertices must be meshed; this is accomplished by creating a VertexMesher graph node, applied to vertices V1 and V2. The setup operation on the VertexMesher node creates no new graph nodes, so the setup traversal terminates at the root node. The graph nodes with user-specified nodes are colored cyan, and automatically created nodes are colored magenta. Note that a given node can represent a meshing operation applied to multiple entities in the BREP model. In practice, this greatly simplifies the mesh graph, with no loss of detail in the overall meshing process. The execute phase traverses the graph in the forward direction, generating mesh for the geometric vertices in the first (non-root) node, for geometric edges in the second node, and finally the surface mesh in the third node.

Fig. 1. (a) User specified Trimesh algorithm

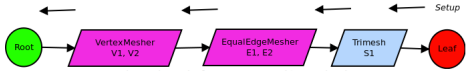


Fig. 1. (b) Setup phase and automatically created nodes

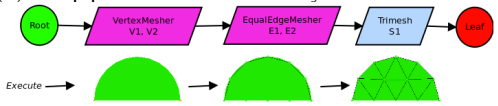


Fig. 1. (c) Final graph with execute phase with model.

In a more complicated example, Fig. 2 shows a users-specified graph that generates a reactor assembly with a boundary layer mesh from scratch. AssyGen operation generates geometry from a text-based input file describing a reactor assembly. This geometry is input to the QuadMesher which feeds into the ExtrudeMesh operation to generate a 3D mesh. Then PostBL, based on user-specified boundary layer thickness, bias etc. generates the desired boundary layer elements for the model. Similar to the previous example, in this example also EdgeMesher and VertexMesher graph nodes are automatically created.

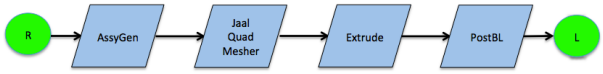


Fig. 2. User specified digraph for creating reactor assembly mesh with boundary layers.

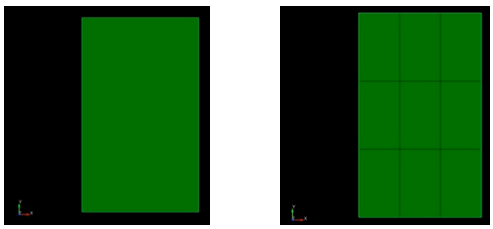
MeshKit contains all basic meshing algorithms required for mesh generation. The goal with MeshKit is to be able to quickly develop algorithms and tools. In a short span of two-three years several new applications and algorithms have developed and published. In the subsequent section, we classify all the algorithms available in MeshKit based on element type. More details and documented examples can be found in MeshKit doxygen page [4].

II.A. Hex/Quad Meshing Algorithms

Hex meshing algorithms available in MeshKit are based on extrusion or sweeping of a quad mesh. Algorithms for supporting hex/quad meshing are listed in this section.

II.A.1 TFI Mapping

Trans-finite interpolation mapping generates an all-quad mesh by transfinite interpolation with i, j parameters. Fig. 3. shows the input geometry and output mesh to TFI mapping algorithm from an example documented in MeshKit code repository.



(a) (b)

Fig. 3. (a) 2D Rectangle geometry, (b) A meshed 2D rectangle.

II.A.2 CAMAL Paver

CAMAL [5] is a proprietary mesh library developed at Sandia National Library. CAMALPaver has been integrated into MeshKit for generating quadrilateral elements using a paving scheme. It must be noted CAMALPaver internally creates an EdgeMesher and VertexMesher graph node that are coded into MeshKit.

II.A.3 JaalQuadMesher

Quad mesher is a MeshKit native algorithm that produces a high quality, isotropic all-quadrilateral meshes for an arbitrary complex surface geometry. Two basic steps used in this algorithm are triangle to quad mesh conversion and global mesh cleanup operation. Specific details and results are published [6].

II.A.4 ExtrudeMesh

ExtrudeMesh is a simple extrusion algorithm that reads in an already meshed 1D or 2D elements and creates a 2D or 3D mesh respectively. It also allows for extrusion to be specified along a rotation path. In-order to produce a mesh that is fit for simulation, material and boundary conditions from the initial mesh are propagated to the final mesh by specifying grouping sets.

II.A.5 Sweeper

OneToOneSweep algorithm generates an all hexahedral mesh by sweeping the source mesh to the target surface. It uses a harmonic function to mesh the target surface with good quality, avoiding expensive smoothing operations. The interior nodes between the source and target surface are generated using cage-based deformation method. Implementation details and results of sweeping algorithm can be found in this year's proceedings of International Meshing Roundtable [7].

II.B. Tet/Tri Meshing Algorithms

MeshKit interfaces with robust automatic Tet/Tri meshing algorithms that have already been developed. This section lists all such the algorithms. Tri-meshing algorithms listed in this section have a restrictive license. Our current work includes incorporation of an open-source algorithm for Tri-meshing.

II.B.1 CAMALTriAdvance, CAMALTetMesher

CAMAL is a proprietary mesh library developed at Sandia National Library. CAMALTriAdvance and CAMALTetMesher are integrated into MeshKit for generating triangular and tetrahedral meshes respectively. It must be noted that both these operations internally create an EdgeMesher and VertexMesher graph node.

II.B.2 NGTetMesher

NetGen [8] is an LGPL licensed automatic tetrahedral mesh generation library. MeshKit has an interface to this library for generating tetrahedral meshes.

II.B.3 TriangleMesher

The Triangle library [9] is widely used for generating exact Delaunay triangulation, constrained Delaunay Triangulation, conforming Delaunay triangulations, Voronoi diagrams and high-quality triangular meshes. It has a restrictive license for commercial use. MeshKit provides an interface for this library for generating 2D triangular meshes.

II.C. Other Algorithms

II.C.1 PostBL

PostBL mesh operation generates boundary layer meshes for an already existing mesh model. Implementation details and results can be found in the proceedings of International Meshing Roundtable 2013 [10]. PostBL can handle triangular, quadrilateral, tetrahedral and hexahedral meshes. Fig. 4 shows a 19-assembly reactor core that needed meshes along the assembly wall and fuel pin boundary for better fluid flow simulations.

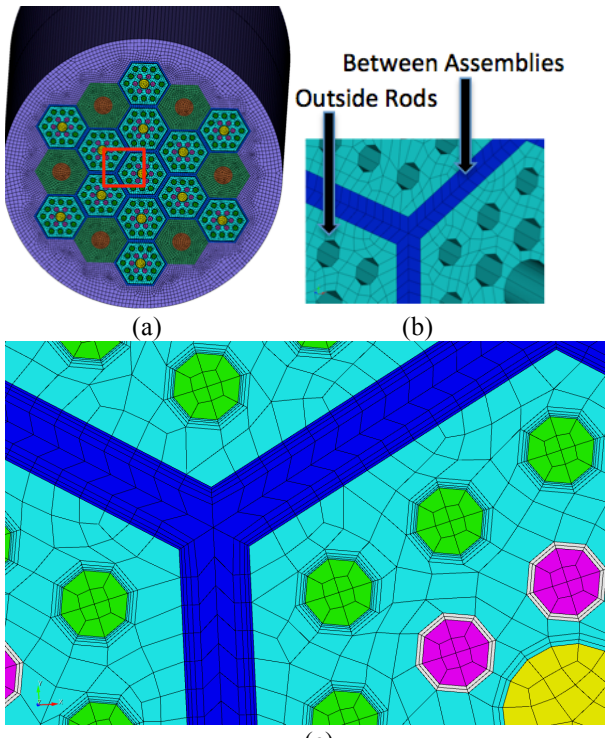


Fig. 4. 19 assembly reactor core mesh: (a) Original mesh. (b) Close-up of original mesh showing fluid and gap regions. (c) Close-up of original mesh showing boundary layers on fluid and gap regions.

II.C.2 Interval matching

Interval assignment is the problem of assigning an integer number of mesh edges to each curve so that the assigned value is close to the goal value, and all containing surfaces and volumes may be meshed independently and compatibly. It is one of the key algorithms for meshing complicated geometries and preventing meshing algorithms to fail from invalid sizing specified by the user. Interval assignment algorithm in MeshKit uses a new optimization function and approach (NLIA), it is found to be faster and more efficient compared to other interval matching algorithms. Integration of this algorithm with other MeshKit algorithms is work in progress. Implementation details and findings of interval matching algorithm can be found in the proceedings of International Meshing Roundtable 2013 [11].

II.C.3 Qslim mesh decimation

Source code of qslim mesh decimation library has been integrated in MeshKit. This algorithm uses edge collapse as a primary simplification method. The cost of each possible edge collapse is established using quadric-based error concept.

II.C.4 Mesquite mesh optimization

Mesquite [12] is an open-source mesh quality optimization package developed at Sandia National Laboratory. It can be used as a standalone operation or it can be coupled with other MeshKit algorithms for mesh quality optimization at the end of mesh generation operation.

II.C.5 MakeWaterTight

MakeWaterTight tool removes gaps, overlaps and discontinuous topology between surfaces. It uses MOAB to read facet based models produced by a solid modeling engine in CGM. It is assumed that the feature size is greater than the facet tolerance, and the facet tolerance is greater than the merge tolerance. Faceted surfaces are skinned to resolve their boundary. Bounding edges of each faceted surface are assembled into loops. Loops are cut into arcs that correspond to faceted curves, using geometric vertices. Each arc is then sealed to its corresponding curve by using node-node and node-edge contraction. The result is a watertight model in which adjacent surfaces share the same faceted edges. Implementation details can be found in an International Meshing Roundtable publication [13].

II.C.6 Structured Block Mesher

This mesh operation generates a simple rectangular structured mesh, sized to completely surround the input model entities using a geometric entities bounding box. Options for grid size, mesh representation and axis type are defined for providing more control over the mesh generation process.

II.C.7 Embedded Boundary Mesher

EBMesh tool can generate Cartesian meshes for solvers that use embedded boundary algorithms. It uses ray-tracing technique based on hierarchical Oriented Bounding Box (OBBs) in MOAB. Each mesh cell is distinguished as being inside, outside or on the boundary of the input geometry, which is determined by firing rays parallel to x/y/z coordinates. EBMesh tool can directly import CAD-based solid model formats and facet-based formats, output from SCDMesh can be also used as input to EBMesh. Boundary cells created by this tool have edge-cut fraction and volume cut fraction information for each material. Detailed explanation, results and comparison with other related tools could be found in an International Meshing Roundtable paper published in 2010 [14].

II.D. Application Specific Algorithms

All the operations listed in this section can be combined and setup as a graph to solve a particular meshing problem. RGG is the nuclear reactor application in MeshKit. Two specific mesh operations implemented for this application are AssyGen and CoreGen. MeshKit also includes three algorithms for generating mesh models of ice sheets. Ice sheets algorithm is composed of a collection of tailored unstructured meshing and mesh-based geometry algorithms in MeshKit. Details of ice sheets algorithms are not presented here.

II.D.1 AssyGen

AssyGen [1] is the first step of the three-step core mesh creation process, it reads an input file describing a reactor assembly lattice and generates an ACIS or OCC – based geometry file. The second step is meshing, after the first step, user may choose to perform meshing using the CUBIT mesh script generated by AssyGen or using meshing algorithms in MeshKit. AssyGen and meshing steps must be performed for each assembly separately.

II.D.2 CoreGen

CoreGen [1] tool reads an input file describing the reactor core arrangement and generates the reactor core mesh or geometry from its component assemblies mesh or geometry files respectively. CoreGen uses CopyMesh, ExtrudeMesh, CopyGeom and MergeMesh algorithms available in MeshKit. A makefile is generated by CoreGen to automate the AssyGen and CoreGen processes for all assemblies that form the reactor core.

III. RESULTS FROM ASSYGEN ALGORITHM

All the models described in this section were run by three physics simulation codes: PROTEUS (neutronics), Diablo (structural mechanics) and Nek5000 (thermo-hydraulics). Nek5000 and Diablo used the same hexahedral mesh. Nek5000 is a spectral element code, so it required hex27 elements. Diablo and PROTEUS are finite element codes that used hex8 elements. Coupled physics simulation results are not presented in this paper.

III.A. XX09 Assembly Mesh

Instrumented XX09 assembly was used in Shutdown Heat Removal Tests (SHRT) that demonstrated passive safety features of the EBR-II Experimental Breeder Reactor. The specific test from which this problem is derived is SHRT17, which is a loss of flow with SCRAM experiment. This test configuration was selected for the multi-physics demonstration because of the availability of temperature and other validation data available from the experiment. Geometry was generated by AssyGen, new feature to model conical pins was developed. Mesh files for all simulation codes are generated by making small modifications to the CUBIT journal files automatically created by AssyGen. For initial coupled simulations Nek5000 and Diablo mesh had 356k hex27 and hex8 elements respectively, whereas, the PROTEUS mesh has 505k hex8 elements. Fig. 5 shows XZ-plane section views of the entire assembly from Inlet (Fig. 5 (i)) to Outlet (Fig. 5 (iv)). Material coloring is as follows: pink: stainless-steel, yellow: sodium, blue: clad, light-blue: fuel, white: bond-sodium and red: fission-gas.

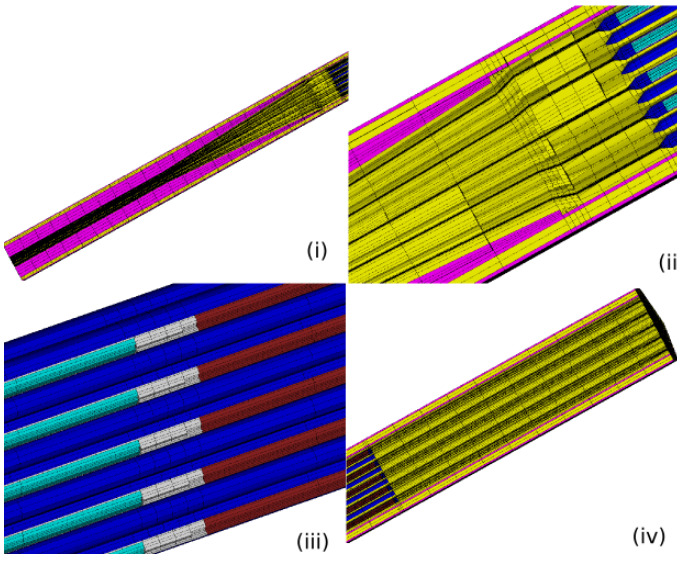


Fig. 5. (i) Cylindrical inlet transitions to form a cone from left to right. (ii) Interface of inlet sodium and pointed conical lower plug, also shows the fuel. (iii) Top of fuel filled with bond sodium and fission gas. (iv) Outlet sodium.

The oblique-sectioned view in Fig. 6 highlights conical fuel pin, bond sodium, clad and stainless steel outer ducts. Elements are observed to have a low aspect ratio, as the mesh of the circular cylinder pins region converges to the pointed conical tip. The number of interval along the height can be adjusted to produce elements with higher aspect ratio elements, for this particular mesh a coarse interval was chosen.

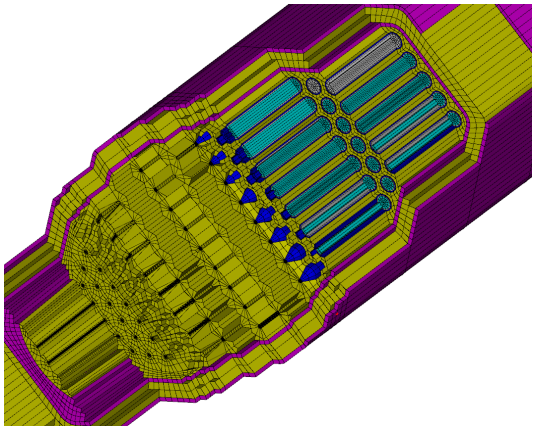


Fig. 6. Oblique sectioned view showing meshing complexities.

This assembly mesh model was run in parallel for performing coupled physics simulations. Partitioning of mesh was performed using the 'mbpart' tool available in MOAB [15]. Fig. 7(i), 7(ii) and 7(iii) show the partitioned mesh with each color representing a processor. Fig. 7(iv) shows the fuel pin and flowmeter arrangement. It must be noted that this arrangement is not visible from the skin of the mesh; a Z-section of the mesh along the fuel region is required to display the pins.

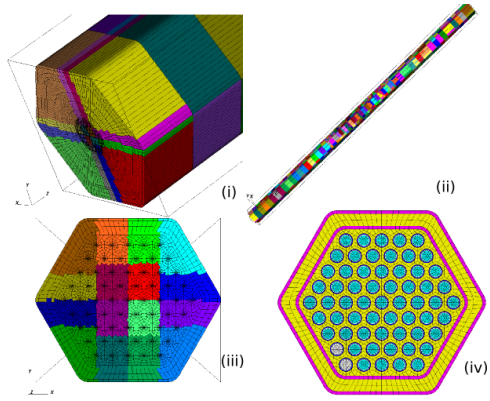
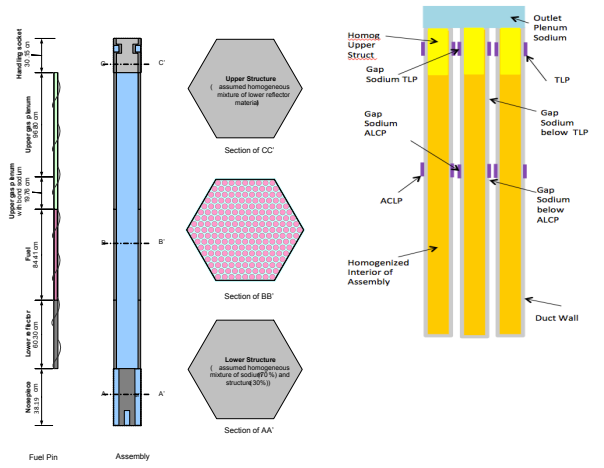


Fig. 7. XX09 assembly (i) Inlet region partitioned mesh (ii) Isometric view of 1024 partitions (iii) Top view of the outlet partition (iv) Top view of mesh showing 61 pins. Mesh is partitioned using the mbpart utility in MOAB. Shows a the sectioned top view of the 61 pin (Note: 59 fuel pins + 2 flowmeters (bottom left)) assembly.

III.B. ABTR Fuel Assembly With Load Pads

AssyGen was used to model the ABTR fuel assembly with load pads and inter-assembly gaps. Load pads are modeled by specifying separate material along the duct at ACLP and TLP locations shown in Fig. 8(b). Inter-assembly gap regions are divided in two equal parts and modeled with every assembly. All the inter-assembly gaps are united when CoreGen assembles the assembly meshes to form to create the core model shown in Fig. 10. Outlet plenums are modeled on top of each assembly for specifying outlet boundary conditions in thermo-hydraulics simulations. The assembly consists of 217 fuel pins. Dimension of one of the pins are shown in Fig. 9. Each pin for a particular axial height consists of four geometric volumes. Hexahedral meshes are obtained by specifying the intervals on the edges, meshing the surfaces and extrusion of the surface mesh. Detailed fuel assembly geometry (non-homogenized, Fig. 8(a)) with varying material specification for a fuel element cross-section consists of over 15k geometric volumes. It takes AssyGen ~20 mins to create the non-homogenized geometric model.



(a) (b)

Fig. 8. (a) Detailed (non-homogenized ABTR fuel assembly). (b) Three homogenized fuel assemblies showing the Above Core Load Pad (ACLP), Top Load Pad (TLP) and outlet plenum regions.

New keywords "NumSuperBlocks" and "SuperBlocks" were introduced to combine material blocks for different physics. "NumSuperBlocks" specifies the number of superblocks and "SuperBlocks" specifies the blocks to be merged to form the new superblock. For example, neutronics models each fuel pin with tens of materials along the height, whereas, thermo-hydraulics consider all the fuel-pins to be one material. These keywords help in creation of one-mesh files that can be used by all physics simulations. Materials and boundary conditions that are not required by a particular physics are ignored. It must be noted that superblocks are not required when modeling meshes for individual physics separately. Time taken to create the homogenized geometry and mesh for this fuel assembly is 2 mins. The mesh has 2.5k hex elements.

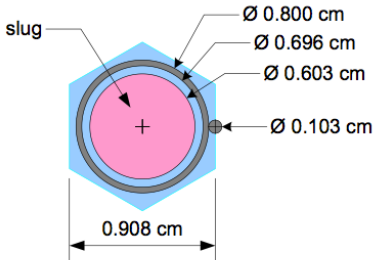


Fig. 9. (a) Section of metal fuel pin showing dimensions in cold condition.

IV. RESULTS FROM COREGEN ALGORITHM

This section presents three different hexagonal reactor models: ABTR, HTGR and EBRII, modeled using the CoreGen tool. ABTR and EBRII reactor core models were used for performing coupled simulations using the SHARP simulation framework [16]. HTGR mesh was used by the PRONGHORN code developed at Idaho National Laboratory [17].

IV.A. ABTR Core Model With Restraint Rings

Detailed configuration with I-J numbering and the number of occurrences of each of the assembly types is shown in Fig. 10. Section III.B highlights one of the fuel assembly models in this reactor. Control, shield, reflector and other assemblies are not shown in this paper. Since, all assemblies have varying properties in axial direction, a common axial configuration must be determined to satisfy all the geometries. This is required, since all the eight assemblies that form this 199-assembly core model must have coincident nodes along the height for creation of a conformal mesh that is fit for simulations.

Homogenized fuel, shield and reflector assemblies are same in-terms of geometry domains, but they have different number of material and boundary condition. Control assembly has an additional duct and sodium layer in each assembly. All 199 assemblies must be modeled as different assemblies with different materials to enable specification of varying densities, inlet/outlet boundary conditions etc. The commonality is 8 assembly mesh files that are arranged in 199 different locations in the core. In-order to overcome the problem of manually creating 199 separate AssyGen files the CreateMatFiles keyword was introduced. This keyword creates a AssyGen files with name "IJ".inp and start material and boundary condition numbers based on "IJ". This enables a numbering scheme that is manageable and helps prescribe temperature and flow-rate etc. for a particular assembly easily.

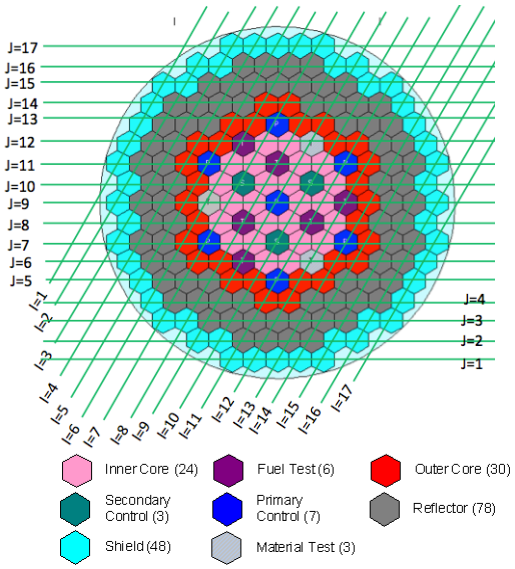


Fig. 10. ABTR core configuration with lines of constant logical I, J assembly regions.

Geometry only CoreGen feature was utilized to create the outer covering with restraint rings shown in Fig. 11. Restraint rings are shown in yellow and red color. All the assembly geometries created by AssyGen are first copy/moved to create a geometric core model. Another separate circular pin (ring) with the same axial divisions as all the other assemblies is created using AssyGen. This pin is subtracted from the geometric core model to create the outer core model and obtain the outer covering geometry shown in Fig. 11. Meshing, material, boundary specification and gap between TLP and ACLP are modeled after this step. The gap is modeled by shrinking the inner surfaces by the gap dimensions. CoreGen is run to create the final homogenized core model from assembly meshes and this outer covering mesh. The final mesh consists of 500k hexahedral elements. Structural mechanics and thermo-hydraulics mesh consists of 1.5k material blocks, whereas, the neutronics mesh consists of 7.5k material blocks. Neutronics mesh requires more material blocks along the height of the fuel pins. It takes less than 5 mins for CoreGen to assemble the core in serial mode.

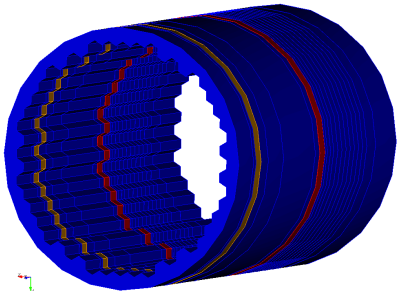


Fig. 11. Outer covering of ABTR core model showing two restraint rings.

IV.B. Homogenized HTGR Core Model

Only 6 input files with 20 lines are needed to generate the High Temperature Gas Reactor (HTGR) model shown in Fig. 12. There are 76 different types of assemblies (based on material) and 331 total assemblies that form this HTGR full core model. The process is completely automatic and modifications to the mesh/geometry are easy once the process is setup using a makefile. It takes 10 mins to create the assembly geometry and mesh from AssyGen input files. CoreGen takes less that one minutes to create the full core model. . A new scheme for radially numbering of materials was introduced because the simulation code that used this mesh was designed to read radially increasing material blocks.

This model contains 864 different fuel materials (right) and 876 materials in total. The complexity and details can be added to the individual assemblies and mesh for other physics can be created using the same setup. Simulation and results obtained from this modeled are presented in a report [16].

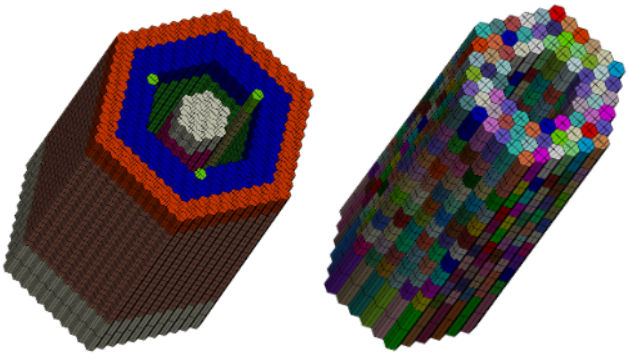


Fig. 12. Homogenized HTGR reactor model showing (a) Other assemblies (b) Fuel assemblies.

IV.C. EBR II Core With XX09 assembly

A homogenized EBRII core mesh with detailed XX09 assembly is shown in Fig. 13. This core model formed of 217 assemblies; the model comprises of 7.9M hex8 elements for neutronics mesh, 4.4M hex8 elements for structural mechanics mesh and 4.4M hex27 elements for thermo-hydraulics mesh. The geometry for XX09 and other homogenized assemblies are generated using AssyGen, then meshing is performed using CUBIT and finally CoreGen creates the resulting EBRII core mesh. For the six assemblies that surround the XX09 assembly, meshing is performed manually by specifying same interval on the edge shared with XX09 assembly. Coarse edge interval is assigned on all other edges.

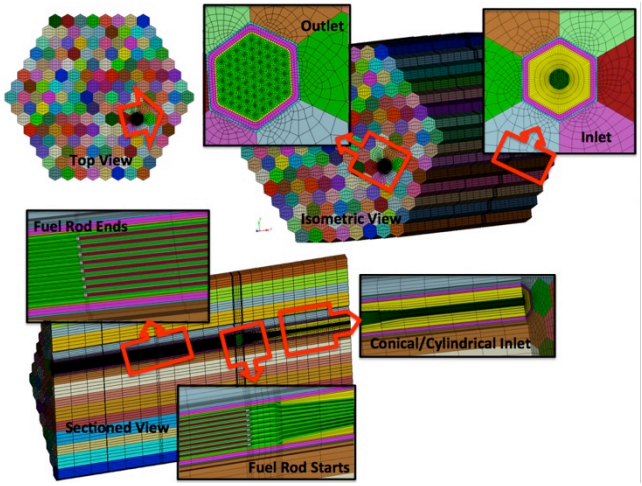


Fig. 13. Homogenized EBRII core with XX09 assembly. Inlet, outlet, isometric and sectioned views are zoomed to show the fuel and other instrumentation pins in the model.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we briefly describe the MeshKit design philosophy, the algorithms available in MeshKitv1.1, and new reactor assembly and core models modeled using the AssyGen and CoreGen tool respectively. New features were introduced and these tools and were used to efficiently and quickly create models of homogenized ABTR, HTGR and EBRII reactors. At present, for the purposes of testing the simulation code, homogenized models were created. Fully detailed, non-homogenized models with fuel, control and other instrumentation pins can be created using RGG.

MeshKit uses a graph-based process for specifying the overall meshing approach, with graph nodes representing meshing and other operations, and graph edges as dependencies between those operations. This approach supports the traditional BREP-driven meshing process found in most other meshing tools, while also enabling a wider variety of meshing processes not strictly based on BREP models. MeshKit contains meshing algorithms for both tri/tet and quad/hex mesh generation. In some cases these algorithms are part of 3rd party meshing tools wrapped by MeshKit, while in other cases the algorithms are implemented directly in MeshKit. The graph-based design enables RGG to interoperate with other meshing algorithms and allow setup of complete reactor mesh generation problem from creation of assembly geometry to core mesh creation in the same program. Due to its unique and modular approach of development, we believe that the development of this library will benefit other user and developer communities outside the nuclear reactor modeling community.

In the near future, we plan to integrate of interval assignment with other algorithms in MeshKit, add more examples to the doxygen-based documentation and formalize CoreGen/AssyGen process with graph-based design in MeshKit. Several new developments in the area of mesh refinement, automatic mesh scheme selection, support for higher order elements and integrating a mesh quality evaluation library are also planned.

ACKNOWLEDGMENTS

We thank the Fathom group at Argonne, who maintains the libraries required by MeshKit. This work was supported in part by the U.S. Department of Energy Office of Nuclear Energy Nuclear Energy Advanced Modeling and Simulation (NEAMS) Program; by the U.S. Department of Energy Scientific Computing Research, Office of Science; and by the U.S. Department of Energy's Scientific Discovery through Advanced Computing program, under Contract DE-AC02-06CH11357.

REFERENCES

1. Tautges, T. J., and Jain, Rajeev, "Creating geometry and mesh models for nuclear reactor core geometries using a lattice hierarchy-based approach". *Engineering with Computers*, *28*(4), 319-329. (2011).

2. Sjaardema, G.D, Tautges, T. J, Wilson, T. J, Owen, S. J, Blacker, T. D, Bohnhoff, W. J, Edwards, T. L, Hipp, J. R, Lober, R. R, and Mitchell, S. A., "CUBIT mesh generation environment, users manual", vol 1. Sandia National Laboratories, Albuquerque. (1994).

3. Jain, Rajeev and Tautges, T. J., "RGG: Reactor Geometry (and Mesh) Generator". *International Congress on the Advances in Nuclear Power Plants, Chicago.* (2012).

4. MeshKit doxygen page: http://www.mcs.anl.gov/~fathom/meshkit-docs/html/index.html

5. CAMAL - The CUBIT Adaptive Meshing Algorithm Library, Sandia National Laboratories, Albuquerque.

6. Verma, C. S., and Tautges, T., "Jaal: Engineering a high quality all-quadrilateral mesh generator". In *Proceedings of the 20th International Meshing Roundtable* (pp. 511-530). (2012).

7. Cai, S., and Tautges, T., (2014). Robust One-to-One Sweeping with Harmonic ST Mapping and Cages: Post-mesh Boundary Layer Generation Tool. In *Proceedings of the 22nd International Meshing Roundtable* (pp. 1-18).

8. Schöberl, J., “NETGEN An advancing front 2D/3D-mesh generator based on abstract rules”. *Computing and visualization in science*, *1*(1), 41-52. (1997).

9. Shewchuk, J. R., “Triangle: Engineering a 2D Quaility Mesh Generator and Delaunay Triangulator”. *Applied Computational Geometry Towards Geometric Engineering.* (pp. 203-222). (1996).

10. Jain, Rajeev, and Tautges, T. J., “PostBL: Post-mesh Boundary Layer Generation Tool”. In *Proceedings of the 22nd International Meshing Roundtable* (pp. 445-464). (2014).

11. Mitchell, S., “Simple and Fast Interval Assignment Using Nonlinear and Piecewise Linear”. In *Proceedings of the 22nd International Meshing Roundtable* (pp. 203-221). (2014).

12. Knupp, P., “Mesh quality improvement for SciDAC applications”. In *Journal of Physics: Conference Series*, *Vol. 46*, No. 1.

13. Smith, B., Wilson, P. and Tautges, T. J., “Sealing Faceted Surfaces to Achieve Watertight CAD Models”. In *Proceedings of the 19th International Meshing Roundtable* (pp. 177-194). (2010).

14. Kim, H., and Tautges, T. J., “EBMesh: An Embedded Boundary Layer Meshing Tool”. In *Proceedings of the 19th International Meshing Roundtable* (pp. 227-242). (2010).

15. Tautges, T. J, Meyers, R, Merkley, K, Stimpson, C, Ernst, C., MOAB: A mesh-oriented database, SAND2004-1592. Sandia National Laboratories, Albuquerque. (2004).

16. Tautges, T.J., Alvaro Caceres, Rajeev Jain, Hong-Jun Kim, Jason A. Kraftcheck, and Brandon M. Smith. "Coupled Multi-Physics Simulation Frameworks for Reactor Simulation: A Bottom-Up Approach." *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering,* (2009).

17. Bingham, A., Ortensi, J., Jain, R., Grindeanu, I., and Tautges, T., SHARP/PRONGHORN Interoperability: Mesh Generation. *NEAMS Report. INL/EXT-12-27171* (2012).